LA-UR-17-20831
Approved for public release; distribution is unlimited.

Title: (U) Introduction to Portage

Author(s): Herring, Angela M.
Certik, Ondrej
Ferenbaugh, Charles Roger
Garimella, Rao Veerabhadra
Jean, Brian Altom
Malone, Christopher M.
Sewell, Christopher Meyer

Intended for: NECDC, 2016-10-10 (Livermore, California, United States)

Issued: 2017-02-03

(U) Introduction to Portage

***Angela Herring*[1], *Ondrej Certik*[2],**
***Charles Ferenbaugh*[3], *Rao Garimella*[4],**
***Brian Jean* [5], *Chris Malone*[6],**
Chris Sewell

[1]angelah@lanl.gov, [2]certik@lanl.gov, [3]cferenba@lanl.gov,
[4]rao@lanl.gov, [5] baj@lanl.gov, [6]cmalone@lanl.gov
Los Alamos National Laboratory, Los Alamos, NM

Abstract

Moving data between geometric representations is a common issue in multi-physics simulations. The Ristra project, which is part of The LANL ATDM program, is developing a new link library, Portage, to assist in both in-line and code-to-code physics linking.

The Portage library is extensible and designed to take advantage of advanced architectures. This will allow the NGC project codes as well as other interested code teams to share remap capability.

1 Introduction

Portage is a hybrid parallel, extensible, intersection-based link library that provides a framework within which one may create custom remap schemes. It is being developed as the remap/link tool for the Ristra project (Ristra is part of the LANL ATDM program).

The current work of the Portage team is focused on producing a general, robust, production quality 2-D and 3-D general polygonal/polyhedral code-to-code link library. The initial version of the library will support but not be fully optimized for Arbitrary Lagrangian Eulerian (ALE) style links. Future versions will optimize Portage for specific remap scenarios.

By beginning with the most general case, the Portage team will provide a link capability within the Ristra code family that can be used from the inception of the Ristra physics codes. The design of Portage will allow later (seamless) specialization and optimization for various code link scenarios.

2 Note: Link Types

Linking is the act of moving physics and geometry data from one mesh representation to another. The key types of links Portage aims to support are code-to-code links, and eventually, ALE style links.

To complete a code-to-code link, data is moved from one mesh representation and code into a new, often topologically different, representation and code. This data movement usually only occurs once within a linked simulation and is usually done to simulate disparate physics regimes.

In an ALE-style link, mesh field data is remapped often throughout the simulation. This remap step can take a considerable portion of the compute time for the simulation. In traditional ALE, the topology does not change. Typical ALE-style remappers (such as swept-face remaps found in FLAG) take advantage of this unchanging topology to reduce the computational time of the remap step.

3 Design

Portage is a modular and extensible library of remap components. These components are designed to be used independently or composed into a custom remapper.

Portage aims to be a central repository for remap components. While the base Portage release will provide a complete, general remap solution, Portage's design is intended to encourage third party extension.

Portage will provide MPI + OpenMP parallelism. Third-party extensions written within the Portage framework are automatically able to take advantage of this paralellism and run effectively on advanced architectures. As third parties create and contribute new remap components, Portage will include the new components as part of the regular releases.

Many remap algorithms of interest can be broken down into three basic operations:

- search–determine a list of cells which may intersect
- intersect–based on candidate cell list, determine which cells intersect and compute relevant moments for use in interpolation
- interpolate–using the computed moments interpolate the data onto the new mesh points

Portage is designed to offer many component options in each category: for instance first and second order interpolators or kd-tree or linear searches. By uncoupling these components from one another we ensure maximum code re-usability (Fig. 1).

Portage is also templated on mesh type–this ensures that codes may use Portage regardless of the underlying mesh data structures. A mesh and state wrapper must be written that provides basic

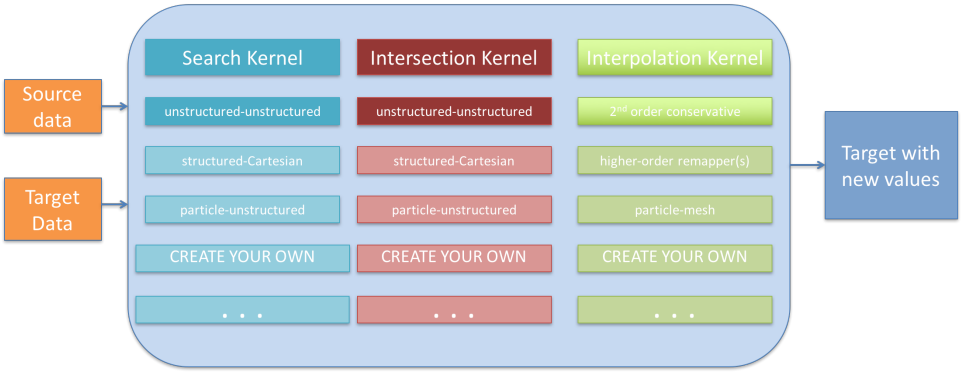

Figure 1: Portage has a modular design. Light colored boxes indicate future capabilities.

information to Portage regarding how to access certain mesh and state components, e.g. how to get nodes, faces, cell data etc. Once this layer is in place, Portage functionality can be used natively.

Once the wrapper is in place, a host code creates a driver specifying the remap for the desired application. For instance if a 3-D 2nd order remap of a general unstructured mesh is desired, the following driver might be composed:

```
Portage::Driver<
        Portage::SearchKDTree, Portage::IntersectR3D,
        Portage::Interpolate_2ndOrder, 3,
        Portage::Jali_State_Wrapper>
          d(sourceMeshWrapper, sourceStateWrapper,
            targetMeshWrapper, targetStateWrapper);
```

This is a complete description of a remapper. The remapper will use the k-d tree search, a general 3-D polyhedral intersection based on r3d [4], and a second order, conservative remap scheme [2].

4 Current Capabilities

The current 1.0 alpha release of Portage contains the tools necessary for general 2-D and 3-D cell and node-centered, single material, conservative remaps and can be found at `http://github.com/laristra/portage`. It includes a serial k-d tree search algorithm, an intersection based on the r3d library of Powell and Abel [4], a conservative first order interpolation component, and a second order conservative interpolation component based on Margolin and Shashkov [2] with Barth and Jesperson limiting schemes [1]. The code is MPI+OpenMP parallel and includes examples, tests, developer documentation, and a quick start guide. The mesh partitioning code is based on work by Plimpton [3].

5 Preliminary Performance Results

Preliminary performance results indicate that Portage performs well in both weak and strong scaling on current architectures. Several scaling studies were run on the ASC capacity machine, snow. The architecture of snow is Intel Broadwell, with two 18-core sockets per node, for a total of 36 cores per node.

All runs are for 2nd order, 3-D, cell-centered remap on Cartesian meshes. Multi-node runs were done using MPI+OpenMP, with one MPI rank per socket and one OpenMP thread per core. For the *standard* runs, both source and target meshes were partitioned over MPI ranks using a simple Cartesian partition; while for the *reverse* runs, the order of ranks on the source mesh was reversed to exercise the MPI communication machinery on a non-optimal case.

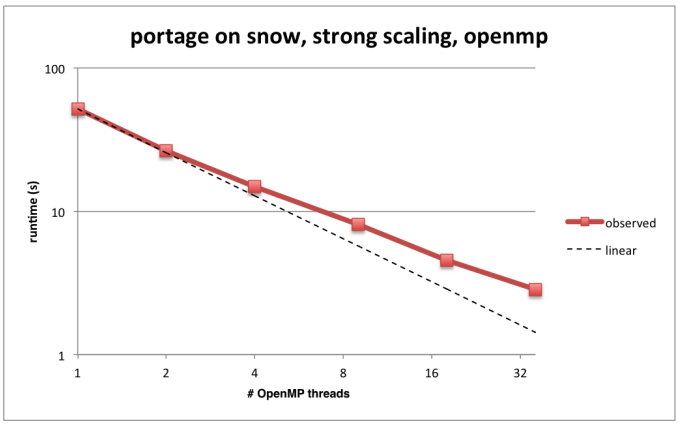

Figure 2: Strong scaling for OpenMP on a single rank, 36x36x36 mesh to a 48x48x48 mesh.

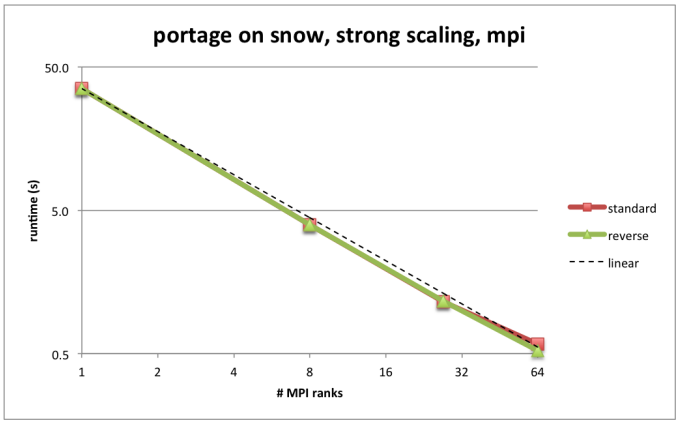

Figure 3: Strong scaling for MPI+OpenMP with one MPI rank per socket, 72x72x72 mesh to a 96x96x96 mesh.

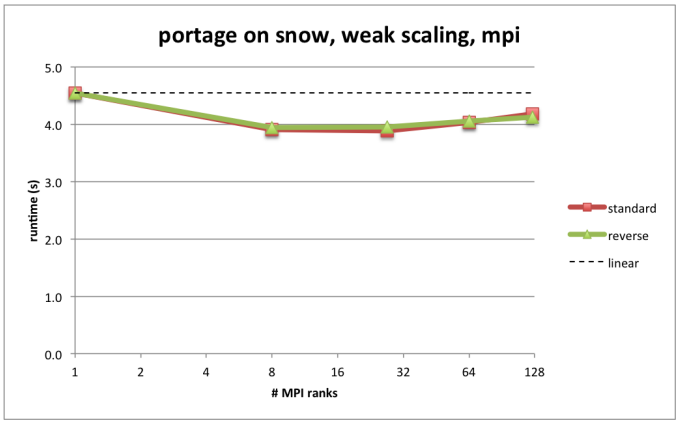

Figure 4: Weak scaling for MPI+OpenMP with one MPI rank per socket, 36x36x36 mesh to a 48x48x48 mesh per MPI rank.

6 Can an intersection-based algorithm compete with a swept-face algorithm?

As Portage will eventually target ALE-style remaps, an initial comparison of the Portage exact intersection to FLAG's swept-face algorithm (SFA) remap was completed. This study is not meant to be rigorous but to give an idea of acceptable remap performance from a user perspective.

Two problems were chosen for the comparison: a favorable problem for the SFA and a difficult problem for SFA. In both cases Portage conducted an exact intersection remap over all cells for an unstructured polygonal/polyhedral mesh. Also, since FLAG currently runs MPI-only (no OpenMP support), both Portage and FLAG were run on a single node in the most advantageous way for each: Portage ran in OpenMP mode (no MPI), while FLAG ran with 16 MPI ranks on a single node.

The first test problem is a contrived problem that starts with a mesh that is slightly distorted. There is no velocity field, but FLAG's "average" relaxer deems some nodes in need of relaxation to get a "better" mesh. The amount of relaxation is small, so SFA does not need to subcycle. In 2-D Portage is within 4x of the FLAG algorithm, but in 3-D it is 11x slower (table I).

Table I: Example 1

Code	2-D 300x300 (s)	3-D 44x44x44 (s)
FLAG (Moonlight - 1 node, 16 MPI ranks)	.132	.397
Portage (Moonlight - 1 node, 1MPI rank, 16 OpenMP threads)	.519	4.303

In the second test problem, an initial rotational velocity field on a 300x300 box was prescribed, with fixed boundaries which causes the internal zones to twist. At cycle 50, ALE was turned on for a single cycle using an Eulerian relaxer. This relaxer attempts to take the mesh back to the initial (t = 0) mesh configuration. The mesh drastically moves upon remap and the SFA needs to subcycle. In this extreme case, in 2-D Portage is actually faster than the SFA, and the 3-D case is only 2.7x slower (Table II).

Table II: Example 2

Code	2-D 300x300 (s)	3-D 44x44x44 (s)
FLAG (Moonlight - 1 node, 16 MPI ranks)	.660	1.854
Portage (Moonlight 1 node, 16 OpenMP threads)	.238	5.01

These results are an encouraging starting place for Portage. Future Portage specialized remap components may be able to take advantage of extra knowledge (such as cells which do not need remapping) and create a drastic reduction in overal remap cost. Turning on OpenMPI + OpenMP would likely further reduce the performance gap.

7 Future Work

Future releases of Portage will include material interface reconstruction, particle-mesh remaps, specialized components for specific remap cases (such as AMR-style remaps), and third party component contributions to Portage as well as improvements to existing algorithms and components.

8 Acknowledgments

We acknowledge the work of Devon Powell of Stanford University and Davis Herring of Los Alamos National Laboratory.

References

[1] Barth, T.J. and Jespersen, D.C. “The design and application of upwind schemes on unstructured meshes,” *27th Aerospace Sciences Meeting*, Reno, NV, 1989.

[2] Margolin, L. and Shashkov, M.J. "Second-order sign preserving conservative interpolation (remapping) on general grids," *Journal of Computational Physics*, v184 n1 pp. 266-298, (2002).

[3] Plimpton, S., Hendrickson, B., and Stewart, J. "A Parallel Rendezvous Algorithm for Interpolation Between Multiple Grids," *Proceedings of the 1998 ACM IEEE SC98 Conference*, Orlando, FL, 1998.

[4] Powell, D. and Abel, T. "An exact general remeshing scheme applied to physically conservative voxelization," *Journal of Computational Physics*, v 297, pp. 340-356, 2015.